Hands On Lab: Map Reduce WordCount program

**Aim:**The aim of this exercise is to learn how to begin creating MapReduce programs using the clouderahadoop Quickstart vm CDH 5.3.Building an initial word-counting program and running it on your local machine.

**QuickStart VM Administrative Information:**

In most cases, the QuickStart VM requires no administration beyond managing the installed products and services. In the event that additional administration is required or that problems occur, this page provides information on accounts and possible explanations and solutions to some common problems

**Accounts**

Once you launch the VM(In previous lab we have launched the CDH 5.3, you are automatically logged in as the cloudera user. The account details are:

username: cloudera

password: cloudera

The cloudera account has sudo privileges in the VM. The root account password is cloudera.

The root MySQL password (and the password for other MySQL user accounts) is also cloudera.

Hue and Cloudera Manager use the same credentials.

**Procedure for WordCount Program Creation and Running:**

Step 1: Open the VMware Workstation from desktop in your computer. Double click on vmware workstation logo as shown below.

**Step 2:**Click on Open a Virtual Machine.

**Step 3:** Choose the cloudera Quickstart vm from the specified folder which you extracted previously and Click Open button as shown in screen below.

**Step 4:**Click on “Power on this virtual machine “and then wait until Cent OS booting will complete (this will take few minutes).

**Step 5:** Minimize the Mozilla Firefox browser and open the pre-installed eclipse IDE from guest Cent OS by double clicking the eclipse icon on desktop which is shown below.

Step 6: Eclipse IDE will open it will take little time.

**Step 7:** Maximize the eclipse window if it is small. Create new project go to Workspace in eclipse (Below Package Explorer) Right click New->Java-> project.

**Step 8:**You will get “New Java Project”window give “Word Count Job” in project name field (you can give any name relevant to your project) then Click on Finish button as shown below.

**Step 9:**“Word Count Job” Project package is created in eclipse workspace. Click arrow button at left side of project name to see the source code and added libraries.

**Step 10:**Adding Jar files

i.Right click on Project name->Build Path->Configure Build Path as shown below.

**ii.** Click on Libraries menu in properties for WordCount window then click on “Add External jars” button.

**iii.** Click on FileSystem at left side in JAR Selection window then enter in “usr” folder->lib

->hadoopfolder one by one.

**IV.** Select all the .jar files in hadoop folder and click on Ok button as shown below.

**V.** Again add some more jar files. Click on “Add External jars” select all jar files from hadoop-0.20-mapreduce path is as follows**“/usr/lib/hadoop-.20-mapreduce”**click ok click ok button.

**Vi.** You will get Reference Library in eclipse workspace and click small arrow at left side to see all jar files as shown in screen shot below.

**Step 11:**Creating class files

**i:**  We need to create three class files such as Driver code, Mapper code and Reducer Code. Right click on src folder below to WordCount project folder in eclipse workspace.

Goto New->Click on class as shown below.

**ii.** You will see “New Java Class” window, there type “WordCount” in Name field then click

Finish button.You can see in below screen.

**iii.** You can see class files in eclipse workspace as shown below.

**iv.**  Similarly create two more classes with WordMapper, WordReducer class names by following previous step as shown below.

**Step 12:** Write Driver code,Mapper code and Reducer code in respective java classes for WordCoundJob in Java Programming by importing all required packages as shown below screen shot.

**Step 13:**Creating Jar file for running word count job.

We need to create jar file for running WordCount job as mapreduce processing.

**i.**Right click on project name move on Export option and click it which is shown below.

**ii.**  You will get export window there expand java folder and then click on “JAR file” option then click Next button as shown below.

**iii.** You will see the JAR Export window there type wc.jar ( jar file name) in “JAR file” field then click “Finish” button as shown below.

**iv.**  Jar file (wc.jar) will be created in home directory workspace by default i.e /home/cloudera/wordspace as shown below.

**Step 14:**Running Word Count Job

This is the final step before running we need to create sample input file in our local system then we need to send input file to HDFS from local system.

**i.**Creating sample Input file in local file system from terminal

**Cmd:**cat > <filename>

**Ex:**cat > file1;

Screen shot shows in below

Note:Local System default home directory is /home/cloudera

**ii.** After creating sample input file in localsystem we need to send input file to HDFS. Sending file to HDFS

**Cmd:**hadoopfs–put <input file name> <HDFS file name>

**Ex:**hadoopfs–put file1 hdfsfile1Note:HDFS default location: /user/cloudera Screen shot shows below.

Browser View screen shot

**iii.**  Running WordCount Job

**Cmd:**hadoop jar <jar name><DriverCode Class Name><Inputfilename From HDFS><OutputDir.name>

**Ex:**hadoop jar wc.jar WordCounthdfs

**Note:**Output file in HDFS always generated in Directory.

**Output:**Word Count Job output from HDFS output directory in Terminal View.

**Output:**Word Count Job output from HDFS output directory in Browser View.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Word count job code\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

import org.apache.hadoop.conf.Configured;  
import org.apache.hadoop.fs.Path;  
import org.apache.hadoop.io.IntWritable;  
import org.apache.hadoop.io.Text;  
import org.apache.hadoop.mapred.FileInputFormat;  
import org.apache.hadoop.mapred.FileOutputFormat;  
import org.apache.hadoop.mapred.JobClient;  
import org.apache.hadoop.mapred.JobConf;  
import org.apache.hadoop.util.Tool;  
import org.apache.hadoop.util.ToolRunner;

public class WordCount extends Configured implements Tool   
{  
public int run(String[] args) throws Exception {  
if(args.length<2)  
{  
System.out.println("please give input and output directories properly");  
return -1;  
}  
  
    JobConf conf=new JobConf(WordCount.class);  
FileInputFormat.setInputPaths(conf, new Path(args[0]));  
FileOutputFormat.setOutputPath(conf,new Path(args[1]));  
conf.setMapperClass(WordMapper.class);  
conf.setReducerClass(WordReducer.class);  
conf.setMapOutputKeyClass(Text.class);  
conf.setMapOutputValueClass(IntWritable.class);  
conf.setOutputKeyClass(Text.class);  
conf.setOutputValueClass(IntWritable.class);  
JobClient.runJob(conf);  
return 0;  
}  
public static void main(String args[])throws Exception  
{  
int i = ToolRunner.run(new WordCount(),args);  
System.exit(i);  
}  
}  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*mappercode\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
import java.io.IOException;  
import org.apache.hadoop.io.IntWritable;  
import org.apache.hadoop.io.LongWritable;  
import org.apache.hadoop.io.Text;  
import org.apache.hadoop.mapred.MapReduceBase;  
import org.apache.hadoop.mapred.Mapper;  
import org.apache.hadoop.mapred.OutputCollector;  
import org.apache.hadoop.mapred.Reporter;

public class WordMapper extends MapReduceBase implements Mapper<LongWritable,Text,Text,IntWritable>  
{  
public void map(LongWritable key, Text value,  
OutputCollector<Text, IntWritable> output, Reporter r)  
throws IOException {  
String s =value.toString();  
for(String word:s.split(" "))  
{  
if(word.length()>0)  
{  
output.collect(new Text(word),new IntWritable(1));  
}  
}  
}  
}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Reducer code\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

import java.io.IOException;  
import org.apache.hadoop.io.IntWritable;  
import org.apache.hadoop.io.Text;  
import org.apache.hadoop.mapred.OutputCollector;  
import org.apache.hadoop.mapred.MapReduceBase;  
import org.apache.hadoop.mapred.Reducer;  
import org.apache.hadoop.mapred.Reporter;  
import java.util.Iterator;  
public class WordReducer extends MapReduceBase implements Reducer<Text,IntWritable,Text,IntWritable>{  
public void reduce(Text key, Iterator<IntWritable> values,  
OutputCollector<Text, IntWritable> output, Reporter r)  
throws IOException {  
int count =0;  
while(values.hasNext())  
{  
IntWritable i=values.next();  
count = count +i.get();  
}  
output.collect(key, new IntWritable(count));  
}  
}